Exact-Differential Large-Scale Traffic Simulation

Masatoshi Hanai
Tokyo Institute of Technology
/ Durham University
Tokyo, Japan
mhanai@acm.org

Toyotaro Suzumura
IBM T.J. Watson Research Center
/ University College Dublin / JST
Yorktown Heights, New York, USA
suzumura@acm.org

Georgios Theodoropoulos
Durham University
Durham, UK
georgios.theodoropoulos@durham.ac.uk

Kalyan S. Perumalla
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
perumallaks@ornl.gov

ABSTRACT

Analyzing large-scale traffics by simulation needs repeating execution many times with various patterns of scenarios or parameters. Such repeating execution brings about big redundancy because the change from a prior scenario to a later scenario is very minor in most cases, for example, blocking only one of roads or changing the speed limit of several roads. In this paper, we propose a new redundancy reduction technique, called exact-differential simulation, which enables to simulate only changing scenarios in later execution while keeping exactly same results as in the case of whole simulation. The paper consists of two main efforts: (i) a key idea and algorithm of the exact-differential simulation, (ii) a method to build large-scale traffic simulation on the top of the exact-differential simulation. In experiments of Tokyo traffic simulation, the exact-differential simulation shows 7.26 times as much elapsed time improvement in average and 2.26 times improvement even in the worst case as the whole simulation.

Categories and Subject Descriptors

I.6.8 [**Simulation and Modeling**]: Types of Simulation—*parallel, distributed, discrete event*

General Terms

Algorithms, Performance

Keywords

Large-scale traffic simulation; parallel discrete event simulation; redundancy reduction

1. INTRODUCTION

Large-scale microscopic traffic simulation has been a beneficial way to research on areas such as prediction of traffic congestion, planning of urban developments, citizen's behavior in emergencies. Unlike other statistical and mathematical ways, such approaches can give detail analysis results of the individual vehicles and other entities like junctions and roads because it actually emulates the vehicles' behavior and interactions with each other.

To analyse by such simulation, it needs to repeat execution many times with different scenarios and parameters. In Tokyo traffic simulation [13, 14, 16, 6], for example, we need to execute 770K times simulation when we simulate what happens if one of the roads is blocked because there are 770K junctions in Tokyo. When we simulate multiple blocks of the roads, we need to execute 2^{770K} times (the sum of combination from 770K choosing 0 to 770K). Also, it often needs to execute a lot of times for parameter tuning (e.g. road speed limit, a time interval of signals) to imitate a realistic situation.

However, previous simulating methods and simulators have a big overhead and a lot of redundancy when repeating the simulation, especially when very small part of scenarios are changed. For example, if only one of all roads is changed in later repeating execution, almost all of simulating results are same as prior execution and the change affects only small part but we have to simulate whole scenarios from the beginning. The reason why the previous simulators have to simulate whole scenarios from the beginning is that a naive spatial partial way (separating simulation space in advance and only simulating the separated part) brings about inconsistency of simulating results from the whole simulating results because it cannot simulate an influence from outside of the separated part and such influence cannot be fixed in advance.

In this paper, we propose a novel technique to simulate only a part of all scenarios and states keeping exact same simulating results as whole execution, called *exact-differential simulation*. The "exact" implies that the output result will be identical. The "differential" implies that only affected events will be reprocessed.

The main idea of the exact-differential simulation is that, in initial whole execution, the simulator stores all events and intermediate states before reprocessing only changing events in later repeating executions by using the stored events and states. There are 3 main contributions in this paper: (i) we describe a way to store processed events and intermediate states and a way to reuse them. (ii) we illustrate imple-

SIGSIM-PADS'15, June 10–12, 2015, London, United Kingdom.
ISBN 978-1-4503-3583-6/15/06 ...$15.00.
DOI: http://dx.doi.org/10.1145/2769458.2769472 .

mentation of the simulator, which meets requirements of the exact-differential simulation. (iii) we evaluate an efficiency and performance of the exact-differential simulation by Tokyo traffic simulation.

The rest of paper is organized as follows. In Section 2, we give the main idea of the exact-differential simulation. We also discuss about static analysis of performance. In Section 3, we illustrate the system implementation. In Section 4, we show modeling of large-scale traffics. In Section 5, we evaluate the exact-differential simulation with the Tokyo traffic simulation. In Section 6, we illustrate related work to our research before conclude in Section 7.

2. EXACT-DIFFERENTIAL SIMULATION

In this section, we describe the processing flow of the exact-differential simulation and its static performance analysis.

2.1 Processing Flow

The simulation flow basically consists of two parts: initial whole execution and repeating execution. In the initial whole execution, the simulator stores all processed events and intermediate states. After the initial whole execution, the simulator starts the repeating execution from a changing point and reprocesses only affected events using the stored events and states.

2.1.1 Initial Whole Execution

In the initial whole execution, events are processed in the almost same way as the optimistic PDES [10], where unlike the normal optimistic PDES, events, cancel events and stored states are never released by the global synchronization (or *GVT, global virtual time*). Instead, such events, cancel events and states are stored to storage for reusing in later repeating execution. Thus, in full, each LP processes events in parallel with a time-sorted event queue and exchanges new generated events with other LPs. When a LP receives the new event with earlier time stamp than its own local time, the LP rollbacks its local time and sends cancel events to neighbors. In the other case that the new received event is older than its local time, the new event is just inserted to its event queue. Cancel events and states are stored whenever a new event is generated. In the global synchronization, older events, cancel events and states than the global time are stored to storage instead of releasing as usual.

2.1.2 Exact-Differential Simulation in Repeating Execution

In the repeating execution, the simulator, at first, inputs a what-if query, which defines changing time and place (LP) and a query's type: ADD or DELETE. Algorithm 1 shows the what-if query processing. In the case of ADD (line 3 – 7), a new event generated from the ADD query is inserted to the event queue and the local time is rollbacked to the new event's time before cancel events are sent to all affected LPs. In the case of DELETE (line 8 – 13), an old event generated from the DELETE query is removed before a cancel event related to the old event is sent. The local time is rollbacked to the old event's time. And finally, cancel events are send to affected LPs. After finishing processing what-if queries, the simulation starts from the rollbacked time in the same way as the optimistic PDES. In the repeating execution,

Algorithm 1 Query Processing Flow

```
 1: while hasWhatIfQuery() do
 2:   query ← getWhatIfQuery()
 3:   if query.type = ADD then
 4:     newEvent ← query.event
 5:     insert(newEvent)
 6:     rollback(newEvent.time)
 7:     sendCancelsToNeighbors()
 8:   else if query.type = DELETE then
 9:     oldEvent ← query.event
10:     delete(oldEvent)
11:     sendCancel(oldEvent)
12:     rollback(oldEvent.time)
13:     sendCancelsToNeighbors()
14:   end if
15: end while
16: while getGlobalTime() < TIME_TO_FINISH do
17:   reprocess unprocessed events with optimistic PDES
18: end while
```

events, cancel events and states are sometimes required to load from storage unlike the usual optimistic PDES. Algorithm 2 shows a mechanism to load the events, cancel events and states during receiving events. We extend the optimistic PDES as showed in algorithm 2 (line 3 – 24). In the exact-differential simulation, received events from other LPs are once buffered before they are inserted to event queues (line 1). If a new received event has less received time than minimum loaded time, which is initialized as infinity, then the stored events, cancel events and states are loaded from the storage (line 7 – 9) before they are inserted to the queues (line 14 – 21). After that, the minimum loaded time is updated to the new received time (line 22), and then the new received event is inserted to the event queue as usual (line 25).

2.2 Static Analysis

In this part, we illustrate efficiency of the repeating execution compered to a naive way. In the repeating execution, a main factor of its performance is how often redundancy events are skipped to process. To clarify the performance improvement, we first define speed up based on the number of processing events and redundancy reduction rate. After that, we discuss detail on the redundancy reduction rate in the repeating execution.

2.2.1 Speed Up

Let E be a set of events; E_{all} be a set of all events; $t_{sim}(\cdot)$ be execution time of processing events; and $t_{init}(\cdot) = t_{init\_local}(\cdot)+t_{init\_global}$ be execution time of initiation, where $t_{init\_local}(\cdot)$ is initializing events time and t_{init_global} is initializing time including state initialization and other initialization independent of events.

The execution time of whole simulation t_{whole} is represented as following.

$$\begin{aligned} t_{whole} &= t_{sim}(E_{all}) + t_{init}(E_{all}) \\ &= t_{sim}(E_{all}) + t_{init\_local}(E_{all}) + t_{init\_global} \end{aligned}$$

We represent T_{n_diff} as execution time of n times repeating exact-differential simulation.

$$T_{n\_diff} = \sum^{n} t_{sim}(E_{re}) + t'_{init}$$

Algorithm 2 Receive Event Processing Flow

```
1: while receiveEventBuffer.isEmpty() do
2:   newEvent ← receiveEventBuffer.dequeue()
3:   /* Extended Part */
4:   if newEvent.time < store.minLoadedTime then
5:     from ← newEvent.time
6:     to ← store.minLoadedTime
7:     oldEvents ← store.getEvent(from, to)
8:     oldCancels ← store.getCancel(from, to)
9:     oldStates ← store.getState(from, to)
10:    while oldEvents.isEmpty() do
11:      loadedEvent ← oldEvens.dequeue()
12:      eventQueue.insert(loadedEvent)
13:    end while
14:    while oldCancels.isEmpty() do
15:      loadedCancel ← oldCancels.dequeue()
16:      cancelQueue.insert(loadedCancel)
17:    end while
18:    while oldStates.isEmpty() do
19:      loadedState ← oldStates.dequeue()
20:      stateQueue.insert(loadedState)
21:    end while
22:    store.minLoadedTime ← from
23:  end if
24:  /* Extended Part End */
25:  eventQueue.insert(newEvent)
26: end while
```

, where $E_{re}(\subseteq E_{all})$ is a set of reprocessing events and t'_{init} is initialization execution time in the exact-differential simulation. Actually, the global initialization is required to execute only one time. Thus, we can represent t'_{init} as follows.

$$t'_{init} = \sum^{n} t_{init\_local}(E_{re}) + t_{init\_global}$$

As the result, T_{n_diff} is represented as following.

$$\begin{aligned} T_{n\_diff} &= \sum^{n} t_{sim}(E_{re}) + t'_{init} \\ &= \sum^{n} t_{sim}(E_{re}) + \sum^{n} t_{init\_local}(E_{re}) + t_{init\_global} \\ &= \sum^{n} \{t_{sim}(E_{re}) + t_{init\_local}(E_{re})\} + t_{init\_global} \end{aligned}$$

We assume processing and initializing time per event is constant. Then, we can represent the execution time by one event processing time t_{ev} as follows.

$$\begin{aligned} t_{sim}(E) &= |E| \cdot t_{ev} \\ t_{init\_local}(E) &= |E| \cdot t_{init\_ev} \end{aligned}$$

, where $|E|$ is the number of elements in E and $t_{ev} = t_{sim}(\{e\})$, $t_{init\_ev} = t_{init\_local}(\{e\})$ $(e \in E)$.

To simplify the discussion, we also assume the number of whole events (E_{all}) and reprocessing events (E_{re}) is constant even if scenarios are changed. Then, the sum of events processing time is represented as following.

$$\begin{aligned} \sum^{n} t_{sim}(E_{all,re}) &= n \cdot |E_{all,re}| \cdot t_{ev} \\ \sum^{n} t_{init\_local}(E_{all,re}) &= n \cdot |E_{all,re}| \cdot t_{init\_ev} \end{aligned}$$

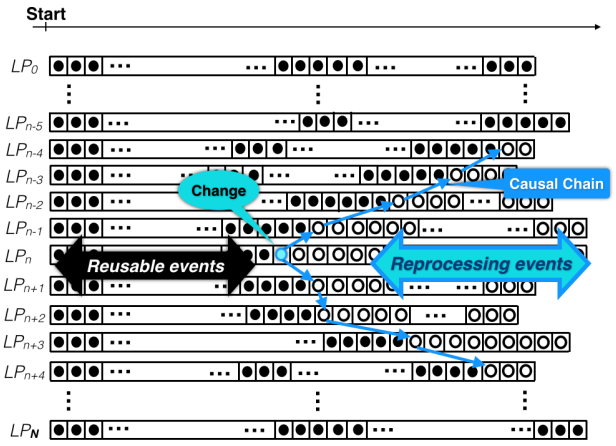

Figure 1: Reusable Events and Reprocessing Events.

We define speed up as the division of naive way's execution time ($T_{n\_whole} := \sum^{n} t_{whole}$) by our proposal one ($T_{n\_diff}$). The speed up is represented as follows.

$$\begin{aligned} (Speed\ Up) &= \frac{T_{n\_whole}}{T_{n\_diff}} \\ &= \frac{\sum^{n}\{t_{sim}(E_{all}) + t_{init\_local}(E_{all}) + t_{init\_global}\}}{\sum^{n}\{t_{sim}(E_{re}) + t_{init\_local}(E_{re})\} + t_{init\_global}} \\ &= \frac{n\{|E_{all}|t_{ev} + |E_{all}|t_{init\_ev} + t_{init\_global}\}}{n\{|E_{re}|t_{ev} + |E_{re}|t_{init\_ev}\} + t_{init\_global}} \\ &= \frac{n\{|E_{all}|t_{ev} + |E_{all}|t_{init\_ev} + t_{init\_global}\}}{n\{r|E_{all}|t_{ev} + r|E_{all}|t_{init\_ev}\} + t_{init\_global}} \\ &= \frac{|E_{all}|t_{ev} + |E_{all}|t_{init\_ev} + t_{init\_global}}{r|E_{all}|t_{ev} + r|E_{all}|t_{init\_ev} + t_{init\_global}/n} \end{aligned}$$

, where r is *redundancy reduction rate* defined as $r := |E_{re}|/|E_{all}|$. In the case that the execution is repeated enough much times, t_{init_global}/n can be ignored.

$$\begin{aligned} (Speed\ Up) &\sim \frac{|E_{all}|t_{ev} + |E_{all}|t_{init\_ev} + t_{init\_global}}{r|E_{all}|t_{ev} + r|E_{all}|t_{init\_ev}} \\ &= \frac{1}{r} \cdot \left(1 + \frac{t_{init\_global}}{|E_{all}|(t_{ev} + t_{init\_ev})}\right) \qquad (*) \end{aligned}$$

This result means that in the case that the global initialization time is much shorter than the event processing and initialization time, or in the case that there are many events to be processed, the speed up is nearly proportional to $1/r$. The more redundancy is reduced, the more speed up it is. On the other hand, in the case that the global initialization cannot be ignored compared to the event processing and initializing time, the speed up depends on the ratio of the global initialization time by the event processing and initializing time, and on the number of all events.

2.2.2 *Redundancy Reduction Rate*

The rest of the section focuses on the redundancy reduction rate, namely parameter r as stated above. Let lp_i be

a logical process and LP be a set of logical processes. We assume there are l logical processes in simulation. The LP is represented as following.

$$LP := \{lp_i | i = 0, 1, 2, ..., l-1\}$$

We define a function that makes a causal chain from one event as $f : E \rightarrow E^l$.

$$f(e) := \{f_0(e), f_1(e), f_2(e), ..., f_{l-1}(e)\}$$

, where $f_i(e)$ is a function from an event to the earliest affecting event in lp_i. Also, let Ei_{re} be an event after $f_i(e)$.

$$Ei_{re} := \{e | e \geq f_i(e)\}$$

The parameter r is represented like that (Figure1).

$$r = \frac{|E_{re}|}{|E_{all}|} = \frac{\sum_{i=0}^{l-1} Ei_{re}}{|E_{all}|}$$

3. IMPLEMENTATION

In this section, we show system implementation. For implementation, there are two requirements to be satisfied. First, our proposal is for actual city-scale or country-scale traffic simulation. Its implementation needs to be scalable to large-scale and to be run in parallel. Second, as we showed in Section 2, our proposal needs to store all processed events and intermediate states in the initial whole simulation. To meet these requirements, our system is designed as the extension of an optimistic parallel traffic simulator.

3.1 Overview

Figure 2 shows the system overview. Our simulator is executed on distributed environment and includes three modules for traffic simulation and one module for storing events and states: application, Time Warp layer, communicator and local storage.

In application, actual simulation logic and algorithms are constructed. The exact-differential simulation mechanism is fully independent of the application code such as modeling of traffic simulation or logic of a vehicles' behavior. The application gets an event and simulation state (namely the state of junction and roads) from Time Warp layer before processing the event based on the vehicles' behavior algorithm, and then returns a new event and a changed state because of the processing event.

The Time Warp layer, which is a core module of our system, consists of a LP manager, LPs and a local storage. Each LP has an event queue, a cancel queue and a state queue just the same way as optimistic PDES. The LPs are managed by a thread pool in LP manager because there are some load imbalances between LPs in the traffic simulation. For example, such load imbalance is happened when in some roads there are a lot of vehicles to be process while in other roads there are few vehicles to be processed. The LP manager also manage the partition of the simulation, that is, the meta-data of each LP. Also, the LP manager controls the access to the local storage, where all processing events and intermediate states are stored in the initial whole execution.

The communicator controls node to node communication using MPI as well as inner process communication between LPs in the same node.

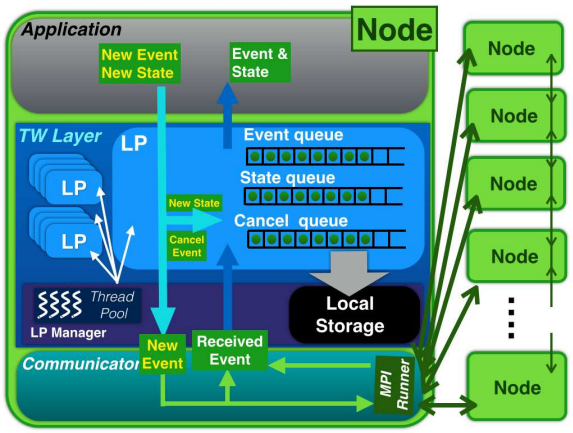

Figure 2: System Overview and Initial Whole Simulation.

3.2 Initial Whole Execution

Figure 2 also shows the flow of initial execution per node. In the initial execution, the simulator at first inputs states and scenarios before simulation. The states data are deployed to each node based on the defined partitioning. After inputting, the simulation starts and processes events (Figure 2). In the initial execution, the LP manager allots a free thread to a LP to process events. The LP passes the earliest unprocessed event to the application before the event is processed in the application. After that, the LP gets a new generated event and a changed state from the application. The new event is sent to a new destination LP via communicator according to a partition discussed later. Also a new cancel event is stored in the cancel queue. If the event has to be sent to other node, the event is communicated via MPI. On the other hand, if the new destination LP is in the same node, the event is sent as inner process communication. Such new sending event is received in the communicator and passed to the destination LP.

Unlike usual optimistic PDES system, events with smaller time stamp than global time are not released after global synchronization. Instead, such events are stored in the local storage for later exact-differential execution.

3.3 Exact-Differential Simulation in Repeating Execution

In repeating exact-differential execution (showed in Figure 3), the system first inputs a changing query and then distributes to a destination LP. The query is received in the LP via communication layer before accessing the local storage to load the changing event and all events affected by the changing event. These events (the changing event and the affected events) are inserted to an event queue to process again. After that, the LP passes the earliest unprocessed event to the application before gets the new generated event and changed state by the application. The new generated event is sent to the destination LP via communicator and the new state is stored to a state queue. After communicator receives the new event, the new event's received time is checked and the affected events are loaded if the time is less than minimum loaded time as discussed in Section 2.

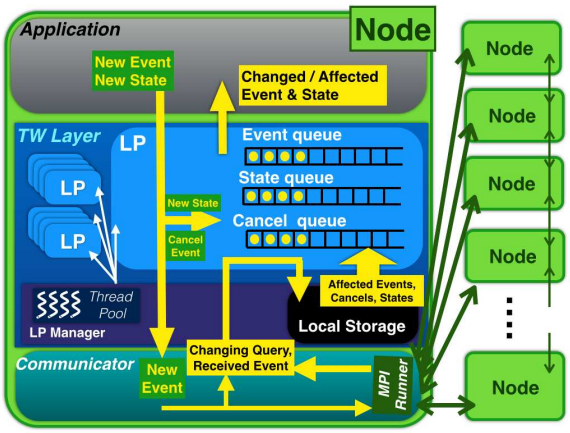

Figure 3: Exact-Differential Simulation.

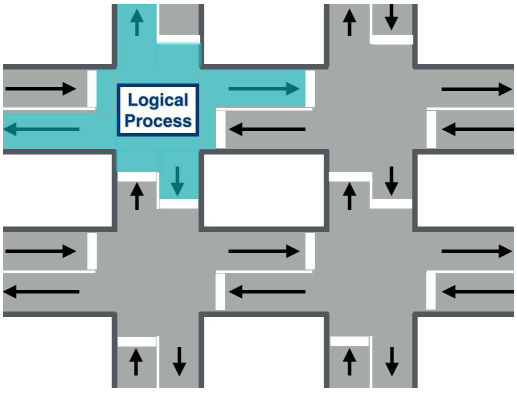

Figure 4: Road Map and a Logical Process Unit.

4. LARGE-SCALE TRAFFIC SIMULATION

In this section, we describe the modeling of traffic system. In a first half of this section, we illustrate how to simulate local vehicles' behavior, that is, how to decide their route at junctions and vehicles' speed on a road. In the other half, we show the model of the global interaction with each other on the road map.

There are two requirement in the modeling way like the system requirement discussed in Section 3. First, the traffic simulation needs to be scalable to city-scale or country-scale. Second, the traffic simulation is required to be modeled on the top of optimistic PDES since the exact-differential simulation is totally based on the optimistic PDES as we discussed in Section 2. To meet such requirement, the model of our traffic simulation is based on IBM Megaffic [13, 14, 17] and SCATTER [15, 18].

4.1 Individual Vehicle's Behavior

The individual vehicle's behavior is based on Megaffic, where it optimises drivers' decisions by estimating some of the parameters of the model from probe-car data before actual simulation execution, differentiating Megaffic from many other traffic simulators which need to calibrate these parameters during the simulation. In short, Megaffic precomputes some of the simulation data, such as, road segments and lanes chosen by the drivers on their route, speed of the vehicles on the road. This is because in large-scale traffic simulation, the processing time of individual vehicles becomes the main bottleneck of the simulation and has to be reduced as much as possible.

In the same way as Megaffic, a vehicle's track of junctions from origin to destination is all fixed before execution by estimation with some defined behavior model, for example, shortest path or minimum hops of junctions. After that, in the execution, the vehicle's speed, traveling time to next junction and selection of a lane are calculated based on some defined behavior models. Finally when the vehicle reaches its destination, it is removed from the simulation.

4.2 Interaction of Vehicles on the Road Map

The global interaction of vehicles around the road map is based on SCATTER. Thus, we mainly use the optimistic PDES technique for parallelization including synchronization. As SCATTER, we represent a vehicle's track as a sequence of events. One event is represented as vehicle's moving from one junction to a next junction. A unit of a logical process in the optimistic PDES is a set of a junction and its outgoing roads (Figure 4). The arrival timing to junctions and synchronization with other junctions are fully controlled by the optimistic PDES.

Also, the road map is partitioned in advance by the k-ways graph partitioning algorithm [12] with METIS [11], which is the software including the k-ways algorithm implementation.

5. EVALUATION

In this section, we describe evaluation of the exact-differential simulation with Tokyo traffic scenarios.

There are two topics to evaluate: efficiency and performance. In the efficiency evaluation, we evaluate how the exact-differential simulation can reduce the number of redundant events in repeating execution. We experiment with two types of changing scenarios. The first one is the case to change vehicle's scenarios. We change a vehicle's track in repeating execution. The second one is the case to change the road map. We change the parameter of one of LPs (one junction and its out-going roads) in repeating execution. The efficiency evaluation shows, in high level, how the exact-differential simulation "potentially" can improve the performance. On the other hand, in performance evaluation, we show the "actual" performance improvement in our implementation, where we evaluate the elapsed time compared to the whole simulation as well as the scalability and parallelization of the simulator.

5.1 Efficiency Evaluation

In this part, we illustrate the efficiency of the exact-differential simulation with Tokyo traffics in 3 hours. Table 1 shows the simulation scenario. We simulate the traffic in Tokyo bay area with 161,364 junctions and 203,363 roads (Figure 5). Based on the Tokyo's statistical data collected by the MLIT (Ministry of Land, Infrastructure, Transport and Tourism) in 2011, totally 5000 vehicles depart from their origin in 3 hours. Each vehicle has a trip pattern which has randomly generated origin/destination. In total, this scenario generates 798,177 events to be outputted, where we use the term "outputted event" as a processed event which is fixed and

never canceled in Time Warp layer. Thus, actually in Time Warp layer, over 798,177 events are handled and canceled.

In this experiment, we use Hamilton4 supercomputer (10 MPI processes) in Durham University but the result of efficiency here is independent of the cluster environment. A result influenced by the environment (thus, elapsed time) is discussed in the performance evaluation.

Road Map	Tokyo Bay Area (Figure 5)
– # of Junctions	161,364
– # of Roads	203,363
Scenario of Tokyo's Traffic	
– Sum of Departing Vehicles	5,000 (3 hours)
– Trips Origin/Destination	Random
Result of Whole Simulation	
– Total Outputted Events	798,177

Table 1: Traffic Scenario.

Figure 5: Road Map of Tokyo Bay Area.

5.1.1 *Evaluation with Changing a Vehicle's Track*

Here, we study the impact of changing vehicles on repeating execution. In this evaluation, we change one of the 5000 vehicles' track paths and then execute the exact-differential simulation. The new track path is generated randomly so that the number of hops is unchanged. We totally change all 5000 vehicles' track path respectively and count the number of events affected by the change, which need to be outputted in the exact-differential simulation.

Figure 8 shows the number of outputted events in the changes. We plot the results in departing time order, but as you can see, we cannot find the impact of departing time in the exact-differential simulation. In Figure 6, we compare the worst case and average case to the whole simulation. In average, the number of outputted events in the exact-differential simulation is 44,261, which is only 5.5 % of the whole simulation. Even in the worst case, the number of outputted events is 233,749, which is 29.2 % of the outputted events in the whole simulation.

5.1.2 *Evaluation with Changing a Road Parameter*

In this evaluation, we show the efficiency of the exact-differential simulation with changing a speed limit parameter of the road map. We change the speed limit of one LP

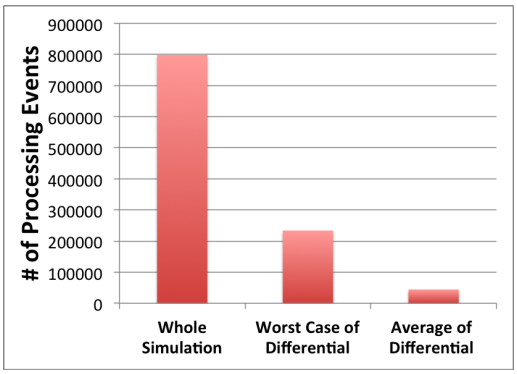

Figure 6: Number of Differential Outputted Events in a Vehicle Change.

and execute the exact-differential simulation from the beginning, where we randomly pick up about 1 % of all 161,364 LPs (1,600 LPs).

Figure 9 shows the number of outputted events in each LP ordered by junction ID, where close numbers are roughly located near points in the actual road map. As you can see, the number of outputted events in the differential simulation has a big gap, that is, some junctions affect the large number of events while the others bring about few events. This is because in Tokyo there are two types of junctions; the first one is the hub, where a lot of vehicles cross over and the other one is a junction, where only few vehicles enter. In Figure 7, we compare the worst case and average case to the whole simulation. In average, the number of outputted event is 61,206. Thus, the events to be outputted are only 7.6 % of whole simulation. Even in the worst case, the number of outputted events in the differential simulation is 297,181, which are 37.2 % of the outputted events in whole simulation.

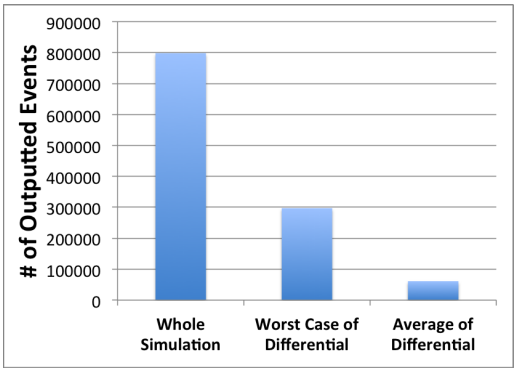

Figure 7: Number of Differential Outputted Events in a Road Change.

To sum up the efficiency evaluation, the exact-differential can reduce over 90% of whole events in average (94.5% in vehicle changes, 92.4% in road changes), and even in the worst case it can reduce over 60% of whole events (70.8 % in a vehicle change, 62.8% in a road change).

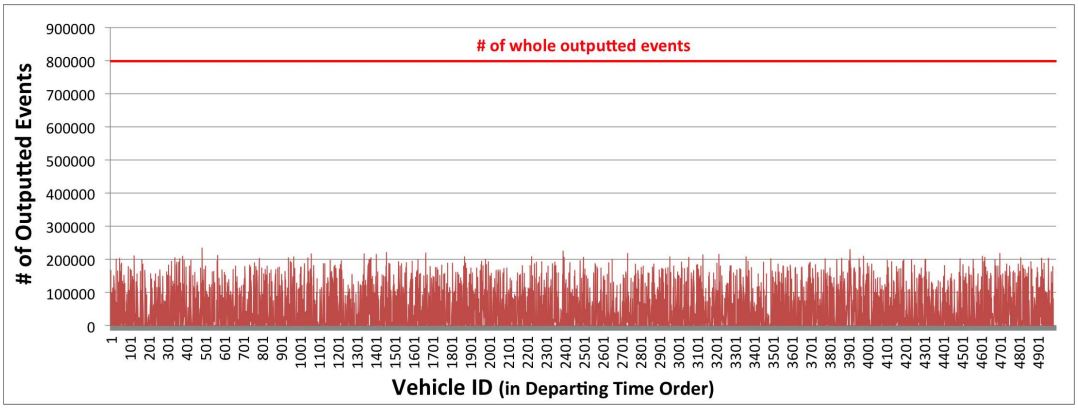

Figure 8: Number of Differential Outputted Events in a Vehicle Change.

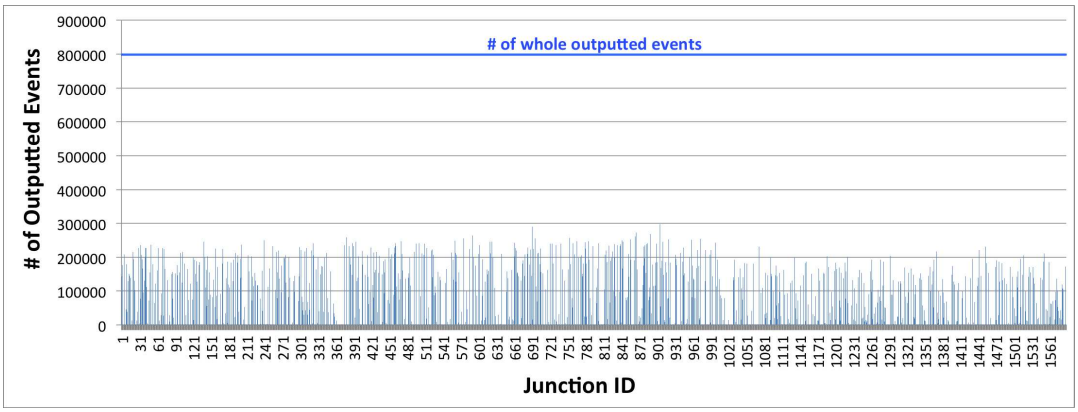

Figure 9: Number of Differential Outputted Events in a Road Change.

5.2 Performance Evaluation

In this part, we show the actual performance of the exact-differential simulation with our simulator. We use the worst and average cases of road changing scenarios as we evaluated above.

Table 2 shows a summary of the evaluation environment. Our simulator is implemented by C++ and MPI with a hybrid parallel architecture, where each MPI process includes multiple threads. We run the simulator with TSUBAME 2.5 Supercomputer in Tokyo Institute of Technology, where there are 12 processors per node with 54GB memory and connected by QDR InfiniBand.

Service	TSUBAME 2.5 in Tokyo Tech.
CPU	Intel Xeon X5670/2.93GHz × 2
Memory	54GB per Node
Network	QDR InfiniBand Interconnect
OS	SLES 11 SP3
MPI	Open MPI 1.6.5

Table 2: Cluster's Configurations.

We change the number of processors according to the table 3. In the simulator, it needs 2 processors per node at minimum because MPI thread and event processing thread are separated in our simulator. Also, it needs 2 MPI process at minimum because of the implementation. Thus, we start from 4 processors and increase the number of cores to 192 (including 16 nodes with 12 threads per node).

Nodes	Threads per Node	Processors
1	4 (2 MPI threads)	4
1	6 (2 MPI threads)	12
2	12 (2 MPI threads)	24
4	12 (4 MPI threads)	48
8	12 (8 MPI threads)	96
16	12 (16 MPI threads)	192

Table 3: Number of Nodes, Threads and Processors.

According to the efficiency result, we pick up the worst case of a road changing scenario needed to be outputted 297,181 events and the average case needed to be outputted 61,530 events, and then evaluate the elapsed time respectively.

Figure 10 shows a strong scaling of the simulator. From 12 processors to 24 processors, the performance becomes once worse because from 24 processors, node to node communication occurs, which is higher cost than a inner node

communication. From 24 processors, the performance becomes better according to the number of processors. In the whole simulation, the elapsed time is improved to be 21.7 % of 4 processors elapsed time. In the exact-differential simulation of the worst case scenario (297,181 outputted events), the elapsed time is improved to be 33.6 % of 4 processors elapsed time. Also, in the exact-differential simulation of the average case scenario (61,530 outputted events), the elapsed time is improved to be 49.7 % of 4 processors elapsed time.

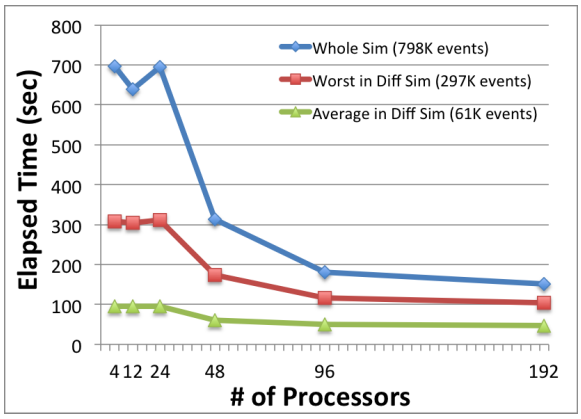

Figure 10: Strong Scaling of Simulation.

Figure 11 shows a speed up from the whole simulation, defined as follows.

$$\frac{(Elapsed\ Time\ in\ Whole\ Sim.\ )}{(Elapsed\ Time\ in\ Exact\text{-}Differential\ Sim.)}$$

The exact-differential simulation achieves at most 7.26 times as much speed up in average case and 2.26 times speed up in the worst case as the whole simulation (4 processors). The speed up decreases according to the number of processors. There are still gaps from ideal cases (showed dotted line in Figure 11), calculated by (*) in Section 2, where we assume $(Speed\ Up) = 1/r$ because $|E_{all}|$ is enough bigger than t_{init_global}. This is because the overhead of the processing events increases according to the number of processors as discussed later.

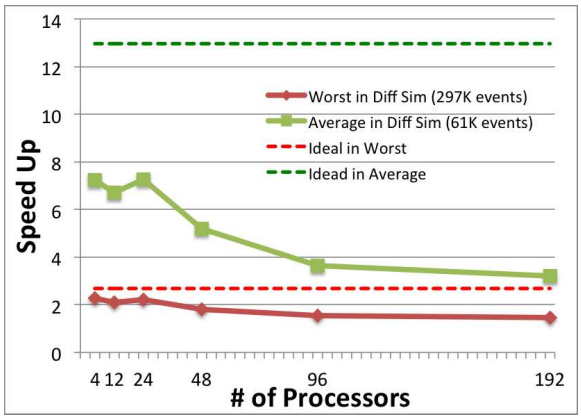

Figure 11: Speed Up from Whole Simulation.

Figure 12 shows elapsed time per outputted event. The elapsed time becomes worse if the number of outputted events decreases in the same number of processors, because the effect of parallel processing decreases in smaller input size.

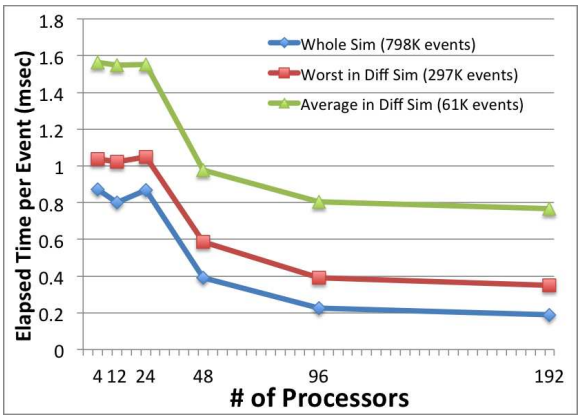

Figure 12: Elapsed Time per Outputted Event.

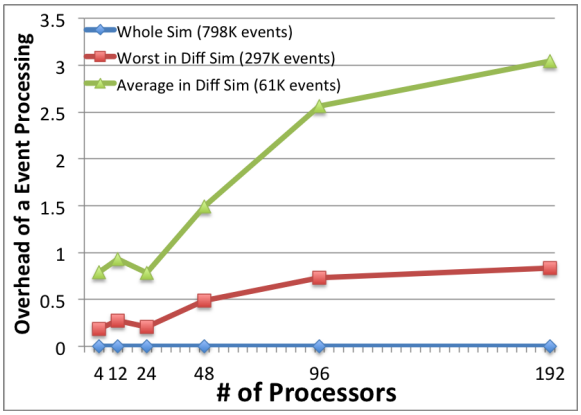

Figure 13: Overhead of Differential Simulation Compared to Whole Simulation.

Figure 13 shows the overhead of repeating execution compared to the whole simulation. The overhead of a differential simulation becomes worse according to the number of processors, where the overhead is defined as follows.

$$\frac{(Time\ per\ Event\ in\ Whole\ Sim) - (Time\ per\ Event\ )}{(Time\ per\ Event\ in\ Whole\ Simulation)}$$

This is because if the more processors there are, the more events are needed to be processed for synchronization, namely, the number of cancel events and the frequency of rollback increase because of the increasing of processors. Also, the overhead increases in the fewer events because the ratio of overhead increases because the elapsed time becomes faster itself in more processors.

To sum up, our simulator with the exact-differential simulation achieves a good performance improvement from the whole simulation. It achieves 7.26 times as much performance improvement in average and 2.26 times improvement even in the worst case as the whole simulation with a road

changing scenario. Also, it achieves a good strong scaling till 48 processors in the worst case. Although the strong scaling in the average case is still limited and it has much overhead compared to the whole simulation, the exact-differential simulation can achieve better elapsed time even with fewer processors than the whole simulation.

6. RELATED WORK

Updateable simulation [5] is highly related in our work. In this research, they propose the technique to simulate a part of events and states in repeating execution by canceling and reprocessing events in the similar way as optimistic PDES technique. In the proposal, they define the *reuse function*, which estimate the number of reprocessing events in repeating simulation and it enables to reprocess part of simulation efficiently. Not only the target domain of application, there are also mainly two differences between the proposal and our approach. First, our approach always achieves the exactly same results as the original whole simulation. This is possible by assuming the separated state (LP) and storing all of intermediate states instead of using the reuse function, which is too general to always ensure the exactly same results. Second, in our proposal, we evaluate the simulator on much larger scale than their proposal. This is one of the main contributions in our research.

Cloning techniques [8, 9, 7, 4, 19] are also ways to reuse the intermediate simulation states and events for efficient repeating simulation. In the cloning of simulation, the simulation states in some decided time is copied and from the decided time the simulation is branched with different parameters or scenarios. The difference from our proposal is that such technique does not have "differential" feature, namely, the cloning technique cannot simulate a part of whole state but they can only simulate wholly from intermediate time.

Reducing the scenario patterns or parameter spaces is the direct and general way to speed up the repeating execution. There have been previous researches with such techniques, illustrated in [1, 2, 3]. For example in [2], the all patterns of scenarios are filtered by random sampling and clustering algorithm before simulation for getting the optimum scenario pattern. In [3], they use GA (genetic algorithm) to pick up the appropriate scenario patterns to be simulated in agent-based simulation.

7. CONCLUSION

In this paper, we proposed the exact-differential simulation for large scale agent-based traffic simulation, which enables to reduce the redundant events in repeating the simulation. In our evaluation, we illustrated how many events are required to be processed in the exact-differential simulation and show a big improvement in reducing the number of processing events. Actually, in the case of vehicles' path changes, we can reduce 94.5 % of processing events in average and even in the worst case, we can reduce 70.8 % of processing events compared to a naive way. In the case to change the parameter of one LP, we can reduce 92.4 % in average and 62.8 % in the worst case. Also with our traffic simulator, we show 7.26 times as much performance improvement in average and 2.26 times improvement even in the worst case as the whole simulation with changing a road's speed limit.

For future works, the implementation should be sophisticated. In Time Warp layer, we should adopt some Time Warp's optimizations like lazy cancellation to reduce the overhead of synchronization. Such optimizations are effective not only to the simulation execution itself but also to the exact-differential simulation since the exact-differential simulation uses directly the mechanism of Time Warp. In local storage layer, to achieve larger traffic simulation, we should expand the size with secondary storage to store vast amount of intermediate state and processed events. Also, we should evaluate various patterns of scenarios with different traffic characteristics to show comprehensive effects of our proposal.

8. ACKNOWLEDGMENTS

This research was partly supported by JST CREST (Japan Science and Technology Agency, Core Research for Evolutional Science and Technology).

9. REFERENCES

[1] S.A. Brueckner and H. Van Dyke Parunak. Resource-aware exploration of the emergent dynamics of simulated systems. In *Proceedings of the 2nd international joint conference on Autonomous agents and multiagent systems (Melbourne, Australia, 14–18 Jul. 2003)*, AAMAS'03, pages 781–788. ACM, 2003.

[2] E. Cabrera, E. Luque, M. Taboada, F. Epelde, and L. Ma Iglesias. Abms optimization for emergency departments. In *Proceedings of the 2012 Winter Simulation Conference (Berlin, Germany, 9–12 Dec. 2012)*, WSC'12, pages 1–12. IEEE, 2012.

[3] B. Calvez and G. Hutzler. Automatic tuning of agent-based models using genetic algorithms. In *Proceedings of the 6th international conference on Multi-Agent-Based Simulation (Utrecht, Netherlands, 25 Jul. 2005)*, MABS'05, pages 41–57. Springer, 2005.

[4] D. Chen, S.J. Turner, W. Cai, B.P. Gan, and M.Y.H. Low. Algorithms for HLA-based distributed simulation cloning. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 15(4):316–345, 2005.

[5] S.L. Ferenci, R.M. Fujimoto, M.H. Ammar, K.S. Perumalla, and G.F. Riley. Updateable simulation of communication networks. In *Proceedings of the 16th ACM/IEEE/SCS Workshop on Parallel and Distributed Simulation (Washington, D.C., USA, 12–15 May 2002)*, PADS'02, pages 107–114. IEEE, 2002.

[6] M. Hanai, T. Suzumura, A. Ventresque, and K. Shudo. An adaptive vm provisioning method for large-scale agent-based traffic simulations on the cloud. In *Proceedings of IEEE 6th International Conference on Cloud Computing Technology and Science (Singapore, 15–18 Dec. 2014)*, CloudCom'14, pages 130–137. IEEE, 2014.

[7] M. Hybinette. Just-in-time cloning. In *Proceedings of the 22nd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (Kufstein, Austria, 16–19 May 2004)*, PADS'04, pages 45–51. IEEE, 2004.

[8] M. Hybinette and R.M. Fujimoto. Cloning parallel simulations. *ACM Transactions on Modeling and*

Computer Simulation (TOMACS), 11(4):378–407, 2001.

[9] M. Hybinette and R.M. Fujimoto. Scalability of parallel simulation cloning. In *Proceedings of the 35th Annual Simulation Symposium (San Deigo, CA, USA, 14–18 Apr. 2002)*, SS'02, pages 275–282. IEEE, 2002.

[10] D.R. Jefferson. Virtual time. *ACM Transaction Programming Languages and Systems (TOPLAS)*, 7(3):404–425, 1985.

[11] G. Karypis and V. Kumar. METIS - a software package for partitioning unstructured graphs, meshes, and computing fill-reducing orderings of sparse matrices-version 5.1.0. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview (Last access data: 10 May 2015).

[12] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graph. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[13] T. Osogami, T. Imamichi, H. Mizuta, T. Morimura, R. Raymond, T. Suzumura, R. Takahashi, and T. Ide. IBM Mega Traffic Simulator. Technical report, Technical Report RT0896, IBM Research–Tokyo, 2012.

[14] T. Osogami, T. Imamichi, H. Mizuta, T. Suzumura, and T. Ide. Toward simulating entire cities with behavioral models of traffic. *IBM Journal of Research and Development*, 57(5):6:1–6:10, 2013.

[15] K.S. Perumalla. A systems approach to scalable transportation network modeling. In *Proceedings of the 2006 Winter Simulation Conference (Monterey, CA, USA, 3–6 Dec. 2006)*, WSC'06, pages 1500–1507. IEEE, 2006.

[16] T. Suzumura and H. Kanezashi. Accelerating large-scale distributed traffic simulation with adaptive synchronization method. In *Proceedings of the 20th ITS World Congress (Tokyo, Japan, 14–18 Oct. 2013)*. ITS Japan, 2013. Paper No.4083.

[17] T. Suzumura, S. Kato, T. Imamichi, M. Takeuchi, H. Kanezashi, T. Ide, and T. Onodera. X10-based massive parallel large-scale traffic flow simulation. In *Proceedings of the 2012 ACM SIGPLAN X10 Workshop (Beijing, China, 11–16 Jun. 2012)*, X10'12, pages 3:1–3:4. ACM, 2012.

[18] S.B. Yoginath and K.S. Perumalla. Parallel vehicular traffic simulation using reverse computation-based optimistic execution. In *Proceedings of the 22nd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (Rome, Italy, 3–6 Jun. 2008)*, PADS'08, pages 33–42. IEEE, 2008.

[19] G. Zhang, M. Fang, M. Qian, and S. Xu. Parallel cloning simulation of flood mitigation operations in the upper-middle reach of huaihe river. In *Proceedings of the 2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (Sanya, China, 10–12 Oct. 2012)*, CyberC'12, pages 73–80. IEEE, 2012.